Python podatkovne strukture, objekti razredov in izjeme

Splošnonamenski programski jezik **Python** je trenutno **najpopularnejši jezik** zaradi enostavne sintakse in obilice prosto-dostopnih programskih knjižnic, dočim pa ni najhitrejši jezik. Jezik se interpretira, kar pomeni, da program `demo.py` v ukazni vrstici izvedemo kot `python demo.py`.

Programming is not science, it is a skill! If you want to run as fast as Usain Bolt, you have to do a lot of running. There is no other way! And it is the same with programming. Just try to run a lot =)

Razredi, objekti in dedovanje razredov

Pri **objektno orientiranem programiranju** skupke podatkov, s katerimi želimo upravljati kot s celoto, združujemo v objekte, ki so določeni z razredi. Pri tem razredi predstavljajo tip objekta, ki združuje atribute, metode in funkcije objekta. V programskem jeziku **Python razrede** definiramo z ukazom `class`. Razredu vedno definiramo konstruktor z metodo `__init__`, ki poskrbi za začetno stanje objekta razreda. Po dogovoru rezervirana beseda `self` predstavlja sam objekt, kar mora biti prvi parameter metode. Navadno definiramo tudi funkcijo `__str__`, ki vrne berljiv opis objekta primeren za izpis uporabniku, funkcijo `__repr__`, ki vrne nedvoumen opis objekta primeren za odkrivanje morebitnih napak s strani programerja, ter morebitne druge metode in funkcije potrebne za delovanje.

Definicija razreda `Animal`, ki naj predstavlja poljubno žival, je prikazana spodaj.

Python

```
class Animal:
  """
  Class representing an animal storing its name, sound and number of legs.
  """
  def __init__(self, name, sound, legs):
    self.name = name
    self.sound = sound
    self.legs = legs

  def __repr__(self):
    return self.__class__.__name__ + "(" + self.name + ", " + self.sound + ", " + str(self.

  def __str__(self):
    return "{:s} '{:s}' with {:d} leg(s) saying `{:s}`".format(self.__class__.__name__, sel

  def get_name(self):
    return self.name

  def get_sound(self):
    return self.sound

  def get_legs(self):
    return self.legs

  def is_wild(self):
    return True
```

Primera objektov razreda `Animal`, ki predstavljata bika `bull` z imenom Ferdinand in stonogo `centipede` z imenom Baltazar, sta prikazana spodaj.

Python

```
>>> bull = Animal('Ferdinand', 'roar', 4)
>>> print(bull)
Animal 'Ferdinand' with 4 leg(s) saying `roar`
>>> print(repr(bull))
Animal(Ferdinand, roar, 4)
>>> print(bull.get_name())
Ferdinand
>>> print(bull.get_sound())
roar
>>> print(bull.get_legs())
4
>>> print(bull.is_wild())
True
>>> centipede = Animal('Baltazar', 'squish', 100)
>>> print(centipede)
Animal 'Baltazar' with 100 leg(s) saying `squish`
>>> print(repr(centipede))
Animal(Baltazar, squish, 100)
```

Ključen koncept pri objektno orientiranem programiranju je **dedovanje razredov**, pri čimer podrazred prevzame vse atribute, metode in funkcije nadrazreda. Podrazredu lahko definiramo dodatne atribute, metode in funkcije, poleg tega pa lahko tudi redefiniramo (tj. prepišemo) funkcionalnosti nadrazreda. Dočim se na metode in funkcije (pod)razreda sklicujemo z rezervirano besedo `self`, se na metode in funkcije nadrazreda sklicujemo z rezervirano funkcijo `super()` (do atributov se vedno sklicujemo z rezervirano besedo `self`).

Primera podrazredov `Cat` in `Dog` razreda `Animal`, ki predstavljata poljubno mačko oziroma psa, sta prikazana spodaj.

Python

```
class Cat(Animal):
  """
  Class representing a cat storing its name, sound and default number of legs.
  """
  def __init__(self, name, sound = 'meow meow'):
    super().__init__(name, sound, 4)

class Dog(Animal):
  """
  Class representing a dog storing its name, sound and default number of legs.
  """
  def __init__(self, name, sound = 'bark bark'):
    super().__init__(name, sound, 4)

  def is_wild(self):
    return False
```

Primera objektov razreda `Cat`, ki predstavljata mački `tom` z imenom Tom in `muri` z imenom Muri, ter primera objektov razreda `Dog`, ki predstavljata psa `bolt` z imenom Bolt in `tacek` z imenom Taček, sta prikazana spodaj.

Python

```
>>> tom = Cat('Tom')
>>> print(tom)
Cat 'Tom' with 4 leg(s) saying `meow meow`
>>> muri = Cat('Muri', 'mjav mjav')
>>> print(muri)
Cat 'Muri' with 4 leg(s) saying `mjav mjav`
>>> bolt = Dog('Bolt')
>>> print(bolt)
Dog 'Bolt' with 4 leg(s) saying `bark bark`
>>> tacek = Dog('Taček', 'hov hov')
>>> print(tacek)
Dog 'Taček' with 4 leg(s) saying `hov hov`
```

Primer podrazreda `Human` razreda `Animal`, ki predstavlja poljubnega človeka, in primera podrazredov `Woman` in `Man` razreda `Human`, ki predstavlja poljubno žensko oziroma moškega, sta prikazana spodaj.

Python

```
class Human(Animal):
  """
  Class representing a human storing its name, sound, default number of legs and list of ch
  """
  def __init__(self, name, sound = 'hey hey'):
    super().__init__(name, sound, 2)
    self.children = []

  def is_wild(self):
    return False

  def get_children(self):
    return self.children

  def add_child(self, child):
    self.children.append(child)

class Woman(Human):
  """
  Class representing a woman storing her name, default sound and number of legs, and list o
  """
  def __init__(self, name):
    super().__init__(name, 'you are wrong')

import random

class Man(Human):
  """
  Class representing a man storing his name, default sound and number of legs, and list of
  """
  def __init__(self, name):
    super().__init__(name, 'I am hungry')

  def is_wild(self):
    return random.choice([True, False])
```

Primer objekta razreda `Woman`, ki predstavlja žensko `mojca` z imenom Mojca, in primer objekta razreda `Man`, ki predstavljata moškega `janez` z imenom Janez, je prikazan spodaj.

Python

```
>>> mojca = Woman('Mojca')
>>> print(mojca)
Woman 'Mojca' with 2 leg(s) saying `you are wrong`
>>> janez = Man('Janez')
>>> print(janez)
Man 'Janez' with 2 leg(s) saying `I am hungry`
```

Statične metode in funkcije razredov

V programskem jeziku Python so **statične metode in funkcije** metode in funkcije razreda, ne objekta razreda. Tako se le-te ne morejo sklicevati na stanje (npr. atribute) objekta z rezervirano besedo `self`. Statične metode in funkcije označimo z dekoratorjem `@staticmethod`.

Python
```
class Human(Animal):
  ...

  @staticmethod
  def get_family(human):
    return [human] + human.get_children()
```

Na statične metode in funkcije se sklicujemo preko razreda, kot je prikazano spodaj.

Python
```
>>> mojca.add_child(janez)
>>> family = Human.get_family(mojca)
>>> print(family)
[Woman(Mojca, you are wrong, 2), Man(Janez, I am hungry, 2)]
```

Posebne metode, funkcije in operatorji objektov

V programskem jeziku Python se **posebne metode, funkcije in operatorji** objekta izvedejo, ko nad objektom uporabimo različne rezervirane besede in ukaze. Sem sodita na primer rezervirana ukaza `str` in `repr` ter operatorji `+`, `-`, `*` ipd. Primer definicije rezerviranega ukaza `len` je prikazan spodaj…

Python
```
class Human(Animal):
  ...

  def __len__(self):
    return len(self.children)
```

…in primer uporabe spodaj.

Python
```
>>> print(len(janez))
0
>>> print(len(mojca))
1
```

Če nad dvema objektoma uporabimo operator `+`, se dejansko kliče rezerviran ukaz `add` prvega objekta. Primer definicije rezerviranega ukaza `add` je prikazan spodaj…

```python
class Human(Animal):
  ...

  def __add__(self, partner):
    couple = Human(self.get_name() + ' & ' + partner.get_name(), 'we are together')
    couple.legs = self.get_legs() + partner.get_legs()
    couple.children = self.get_children() + partner.get_children()
    return couple
```

…in primer uporabe spodaj.

```python
>>> couple = mojca + janez
>>> print(couple)
Human 'Mojca & Janez' with 4 leg(s) saying `we are together`
>>> print(couple.get_children())
[Man(Janez, I am hungry, 2)]
```

Podobno se pri uporabi operatorja `-` kliče rezerviran ukaz **`sub`**, operatorja `*` ukaz **`mul`** in operatorja `/` ukaz **`div`**.

Če nad objektom uporabimo **oglate oklepaje** `[...]`, enako kot pri dostopu do elementov podatkovne zbirke, se dejansko izvede rezerviran ukaz **`getitem`** ali **`setitem`**. Tako lahko naslavljamo, pridobivamo in določamo elemente objekta po indeksu. Primer definicije rezerviranih ukazov `getitem` in `setitem` je prikazan spodaj…

```python
class Human(Animal):
  ...

  def __getitem__(self, i):
    return self.children[i]

  def __setitem__(self, i, child):
    self.children[i] = child
```

…ter primer uporabe spodaj.

```python
>>> print(mojca.get_children())
[Man(Janez, I am hungry, 2)]
>>> print(mojca[0])
Man 'Janez' with 2 leg(s) saying `I am hungry`
>>> mojca[0] = Human('Otrok')
>>> print(mojca[0])
Human 'Otrok' with 2 leg(s) saying `hey hey`
```

Če nad objektom uporabimo **(navadne) oklepaje** `(...)`, enako kot pri klicu metode ali funkcije, se dejansko izvede rezerviran ukaz `call`. Tako lahko objekte razredov uporabljamo enako kot funkcije. Primer definicije rezerviranega ukaza `call` je prikazan spodaj…

```python
class Human(Animal):
  ...

  def __call__(self, human):
    print(self.get_name() + ' likes ' + human.get_name())
```

…in primer uporabe spodaj.

```python
>>> for i in range(len(mojca)):
>>>   print(mojca[i])
>>>   mojca(mojca[i])
Human 'Otrok' with 2 leg(s) saying `hey hey`
Mojca likes Otrok
```

Programske napake, izjeme in lovljenje izjem

V programskem jeziku Python se **izvajanje** programa nemudoma **prekine**, če pride do sintaktične **napake** pri interpretaciji (tj. `Error`) ali nejasne **izjeme** pri izvajanju (tj. `Exception`).

Primer sintaktične napake pri interpretaciji je prikazan spodaj.

```python
>>> if True print('True')
  File "<stdin>", line 1
    if True print('True')
                ^
SyntaxError: invalid syntax
```

Primeri izjem pri izvajanju so prikazani spodaj.

Python

```
>>> x = 0
>>> ...
>>> y = x * z
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'z' is not defined
>>> y = 1 / x
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
>>> y = x + 'foo'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>> y = 'foo' + x
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
```

Lastne izjeme definiramo kot podrazrede nadrazreda `Exception`, kot je prikazano spodaj…

Python

```
class HumanException(Exception):
  def __init__(self, string):
    super().__init__(string)
```

…in jih kličemo ali dvignemo z uporabo rezerviranega ukaza `raise`.

Python

```
class Human(Animal):
  ...

  def __getitem__(self, i):
    if i < 0 or i >= len(self.children):
      raise HumanException('child index out of range')
    return self.children[i]

  def __setitem__(self, i, child):
    if i < 0 or i >= len(self.children):
      raise HumanException('child index out of range')
    elif type(child) is not Human: # not isinstance(child, Human)
      raise HumanException('child must be human')
    self.children[i] = child
```

Kljub temu, da izjema načeloma prekine izvajanje programa, jo lahko ujamemo z uporabo rezerviranega konstrukta `try ... except ...` in se na njo ustrezno odzovemo. V tem primeru se program

normalno izvaja naprej.

Primer lovljenja izjeme je prikaza spodaj.

```Python
>>> print(mojca[0])
Human 'Otrok' with 2 leg(s) saying `hey hey`
>>> try:
>>>   print(mojca[1])
>>> except HumanException as e:
>>>   print(e)
child index out of range
```

Dočim programski jezik Python omogoča, da podrazredi dedujejo več kot en nadrazred, kar pomeni, da ima sin lahko več očetov, to v programskih jezikih kot je Java ni mogoče. Za ta namen so definirani vmesniki (razredov), kar bomo obravnavali v nadaljevanju.